

Algorithmen und Datenstrukturen

[github/bircni](#)

November 11, 2022

Inhaltsverzeichnis

Seite

1 Mathematische Grundlagen

1.1	Reihen	
1.2	Potenzen und Logarithmen	
1.3	Notationskonventionen	
1.4	Grundbegriffe der Graphentheorie	

2 Rekursive Algorithmen

2.1	Prinzip der Rekursion	
2.2	Korrektheit rekursiver Algorithmen	
2.3	Rekursive Berechnung der Potenzmenge	
2.4	Algorithmenprinzip "Backtracking"	

1 Mathematische Grundlagen

1.1 Reihen

Arithmetische Reihe

- **Allgemeine arithmetische Reihe:** $a_0 + (a_0 + d) + (a_0 + 2d) + \dots + (a_0 + n \cdot d)$

$$\sum_{i=0}^n (a_0 + i \cdot d) = (n+1) \left(a_0 + d \frac{n}{2} \right)$$

Beispiel: Summe der ungeraden Zahlen von 1 bis 99, d.h. $1 + 3 + 5 + \dots + 99$:

$$a_0 = 1$$

$$d = 2$$

$$n = 49$$

Ergebnis: $50 \cdot (1 + 2 \cdot 49/2) = 2500$

- **Gaußsche Summenformel:** $1 + 2 + 3 + \dots + n$, also Summe der natürlichen Zahlen von 1 bis n . Dies ist der Spezialfall mit $a_0 = 0$; $d = 1$.

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}$$

Beispiel: Summe der Zahlen von 1 bis 50:

$$50 \cdot 51 / 2 = 1275$$



wichtig

1.2 Potenzen und Logarithmen

Der Logarithmus ist die Inverse der Potenzfunktion. $\log_a(x) = y \iff a^y = x$

spezielle Logarithmen:

$\lg(x) = \log_2(x)$, $lg(x) = \log_{10}(x)$, $\ln(x) = \log_e(x)$

1.3 Notationskonventionen

$\lceil x \rceil$ zur nächsten ganzen Zahl aufrunden

$\lfloor x \rfloor$ zur nächsten ganzen Zahl abrunden

$[a..b] = x | a \leq x \wedge x \leq b$ mit Intervallgrenzen

$]a..b[= x | a < x \wedge x < b$ ohne Intervallgrenzen

$arr[i..k]$ Teilfolge der Elemente von $arr[i]$ bis $arr[k]$

1.4 Grundbegriffe der Graphentheorie

Graphen bestehen aus einer Menge von Knoten und Kanten, die diese verbinden.

Ein Graph ist gerichtet, wenn die Kanten eine Richtung haben.

Für einen Knoten v eines gerichteten Graphen $G = (V, E)$ ist der Eingangsgrad $\text{indeg}(v)$ die Anzahl der Kanten, die in v enden, und der Ausgangsgrad $\text{outdeg}(v)$ die Anzahl der Kanten, die von v ausgehen.

Ein Zyklus ist ein Weg der bei einem Knoten startet und endet.

Ein gerichteter Graph ist zusammenhängend, wenn es einen Weg zwischen jedem Knotenpaar gibt.

Ein Baum hat einen Knoten als Wurzel, jeder Knoten hat genau einen Vorgänger und ist zusammenhängend.

Ein Knoten ohne Kinder heißt Blatt. Ein leerer Baum hat die Höhe 0. Ein Binärbaum ist ein Baum, dessen Knoten maximal zwei Kinder haben.

Traversierungen: Preorder (WLR), Inorder (LWR), Postorder (LRW)

2 Rekursive Algorithmen

2.1 Prinzip der Rekursion

Ein rekursiver Algorithmus besteht aus einem Basisfall und einem rekursiven Aufruf. Der rekursive Aufruf muss immer kleiner werden, damit die Rekursion endet. Die Rekursion kann durch eine Schleife ersetzt werden.

```
public static double sum_v2(double[] arr) {
    return sum_v2(arr, 0, arr.length-1);
}
/** Berechnet Summe der Werte von arr[firstIndex..lastIndex] */
private static double sum_v2(double[] arr, int firstIndex, int lastIndex) {
    if (firstIndex == lastIndex) {
        // zu summierender Bereich besteht nur aus einem Element
        return arr[firstIndex];
    }
    else {
        int mid = (firstIndex + lastIndex) / 2;
        return sum_v2(arr, firstIndex, mid) + sum_v2(arr, mid+1, lastIndex);
    }
}
```

2.2 Korrektheit rekursiver Algorithmen

Ein Beweisverfahren ist die Berechnungsinduktion.

Beweis mittels Berechnungsinduktion. Eigenschaft, die nachgewiesen werden soll:

$$P((x,n),y) :\Leftrightarrow x^n = y$$

d.h. wir wollen zeigen, dass $h(x,n) = x^n$ gilt.

- **Induktionsbasis** (Argumente führen zu keinem rekursiven Aufruf)

Fall $n = 0$:

Ergebnis: $h(x, n) = 1 = x^0$, d.h. Eigenschaft erfüllt

- **Induktionsschritte** (Argumente führen zu rekursiven Aufrufen):

Wir können als Induktionshypothese für die rekursiven Aufrufe verwenden, dass $h(z_i, k_i) = z_i^{k_i}$ gilt.

Fall $n > 0$, n gerade:

Ergebnis laut Programmcode: $h(x,n) = y*y$, wobei $y = h(x, n/2)$

Wir verwenden die Induktionshypothese: es gilt $h(x, n/2) = x^{n/2}$

somit $h(x,n) = y*y = x^{n/2} * x^{n/2} = x^n$,

d.h. Eigenschaft in diesem Fall auch für n erfüllt.

Fall $n > 0$, n ungerade:

Ergebnis laut Programmcode: $h(x,n) = x * h(x,n-1)$

Wir verwenden die Induktionshypothese: es gilt $h(x, n-1) = x^{n-1}$

Somit $h(x,n) = x * h(x,n-1) = x * x^{n-1} = x^n$,

d.h. Eigenschaft in diesem Fall auch für n erfüllt

- **Induktionsschluss:** somit folgt für alle $n \geq 0$, dass $h(x,n) = x^n$ gilt.

2.3 Rekursive Berechnung der Potenzmenge

Beispiel:

Menge: $M = \{a, b, c\}$

Potenzmenge: $\rho(M) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$

2.3.1 Rekursiver Lösungsansatz

a) in welchen einfachen Fällen kann die Lösung direkt angegeben werden?

der einfachste Fall ist die leere Menge $M = \emptyset$

die leere Menge hat nur sich als Teilmenge $\rho(\emptyset) = \{\emptyset\}$

b) Wie können in nicht einfachen Fällen die Teilmengen bestimmt werden?

Sei Menge $M = \{a_1, \dots, a_{n-1}, a_n\}$ nicht leer ($n \geq 1$)

1. Wir wählen ein Element der Menge, z.B. a_n

2. Es gibt nun zwei Arten von Teilmengen:

T^+ Teilmengen, die das Element a_n enthalten

T^- Teilmengen, die das Element a_n nicht enthalten

Die Menge aller Teilmengen ist die Vereinigung von T^+ und T^- , d.h. $\rho(M) = T^+ \cup T^-$

Die Menge T^+ kann nun rekursiv berechnet werden, indem wir a_n aus M entfernen und die Potenzmenge von M berechnen.

Die Menge T^- ist die Potenzmenge von M ohne a_n .

Die Potenzmenge von M ist also die Vereinigung von T^+ und T^- .

Beispiel: Wenn $M = \{a, b, c\}$

Wähle z.B. c als Element:

T^- : alle Teilmengen ohne c , also alle Teilmengen von $\{a, b\}$

$T^- = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$

T^+ : alle Teilmengen mit c , Nimm zu jeder Teilmenge von T^- und füge c hinzu

$T^+ = \{\{c\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$

Insgesamt: $\rho(\{M\}) = T^+ \cup T^- = \{\emptyset, \{a\}, \{b\}, \{a, b\}, \{c\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$

2.3.2 Algorithmischer Ansatz

Falls M leer ($M = \emptyset$)

leere Menge ist die einzige Teilmenge

Falls M nicht leer Wähle ein Element a_n aus M

Berechne Sammlung T^- aller Teilmengen von M ohne a_n (rekursiv)

Berechne Sammlung T^+ aller Teilmengen, die a_n enthalten:

Nimm dazu jede Menge aus T^- und bilde eine neue Menge, indem a_n hinzugefügt wird

Die Menge aller Teilmengen ist die Vereinigung von T^+ und T^-

```
private static <E> Set<Set<E>> allSubsets(E[] arr, int maxIndex) {
    Set<Set<E>> resultSet = new HashSet<Set<E>>();
    if (maxIndex >= 0) {
        // Menge ist nicht leer, waehle letztes Element im gegebenen Bereich
        E selected = arr[maxIndex];
        // Bilde rekursive alle Teilmengen ohne selected
        Set<Set<E>> resultSet1 = allSubsets(arr, maxIndex - 1);
        // nimm jede dieser Mengen zum Ergebnis hinzu
        resultSet.addAll(resultSet1);
        // bilde alle Teilmengen, die selected enthalten
        for (Set<E> set1 : resultSet1) {
            // Erzeuge Kopie der Menge aus resultSet1 und nimm gewaehltes Element dazu
```

```
Set<E> set2 = new HashSet<E>(set1);
set2.add(selected);
    // fuege die ergaenzte Kopie zum Ergebnis hinzu
resultSet.add(set2);
}
} else {
    // Menge ist leer. Leere Menge hat nur leere Menge als einzige Teilmenge
Set<E> emptySet = new HashSet<>();
resultSet.add(emptySet);
}
return resultSet;
}
```

2.4 Algorithmenprinzip "Backtracking"